

# Reverse Engineering grosser Systeme

Erfahrungsbericht über die Modellbildung  
zu bestehenden grossen Systemen

# Vorstellung

**Thomas Rittel, EADS/LFK**

**Aufgabe: Nachweisführung bei der Entwicklung eines  
Waffenführungssystems**

**Thomas.Rittel@lfk.eads.net**

**Dr. Rudolf Hauber, Kölsch & Altmann GmbH**

**Aufgabe: Methoden- und Technologieberater für OO/UML**

**RHauber@ka-muc.de**

# Ausgangssituation

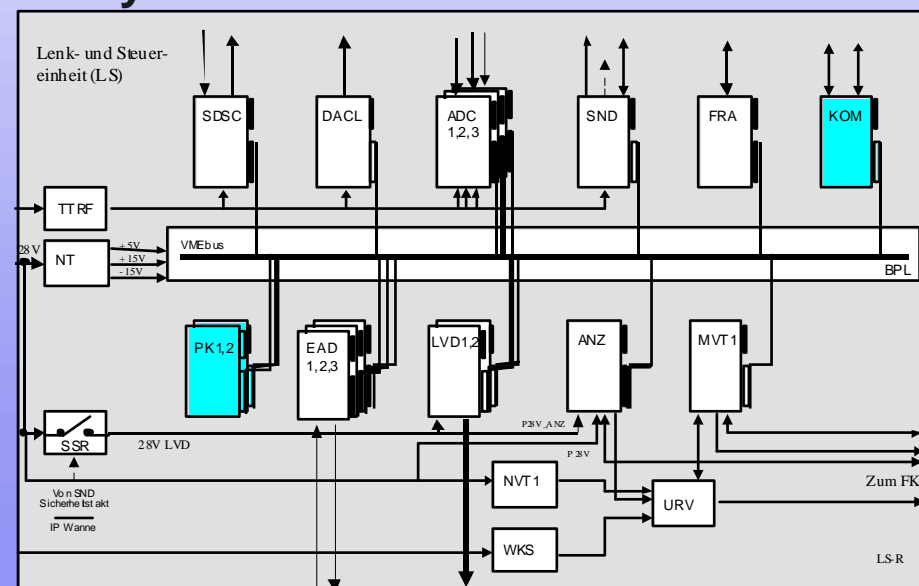
- **Komplexes beim Kunden eingeführtes System**
  - historisch über Jahre gewachsene inkonsistente Dokumentation
  - Know How in den Köpfen weniger „local heroes“
  - Nachweis/Test wurde unter Termin- und Kostendruck vernachlässigt
- **System soll technisch verbessert werden und neue Features beinhalten**
  - => Termine und Kosten sind nicht einzuhalten
  - => Neues Produkt enthält zu viele Fehler, schlechte Qualität
  - => Entwicklungsprozess wird instabil
  - => Entwicklungsstatus wird unübersichtlich

# Lenk-/Steuereinheit für Flugabwehr-System

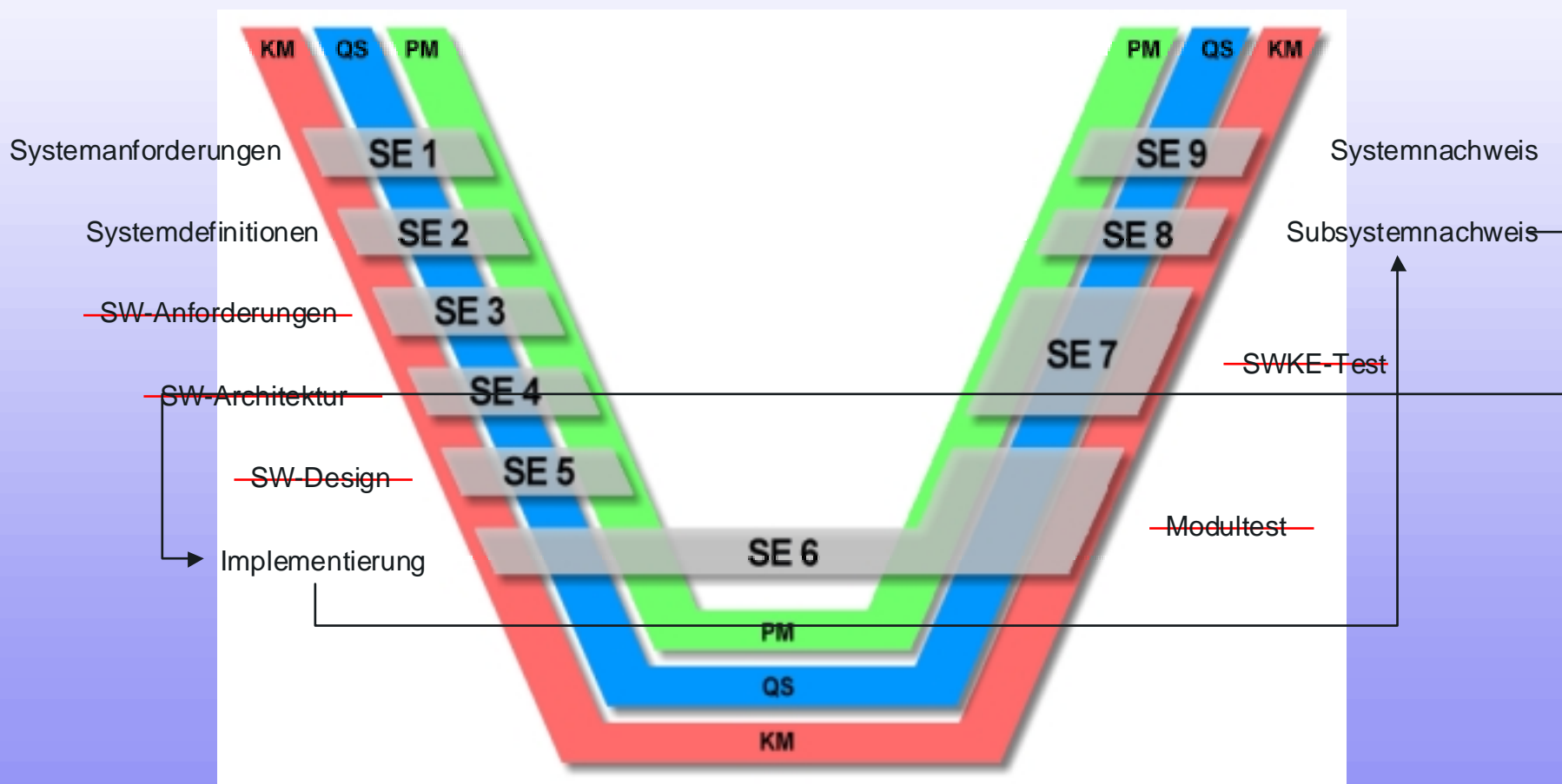
Steuerkommandos



## Systemaufbau Lenk-/Steuereinheit



# Prozess-Assessment



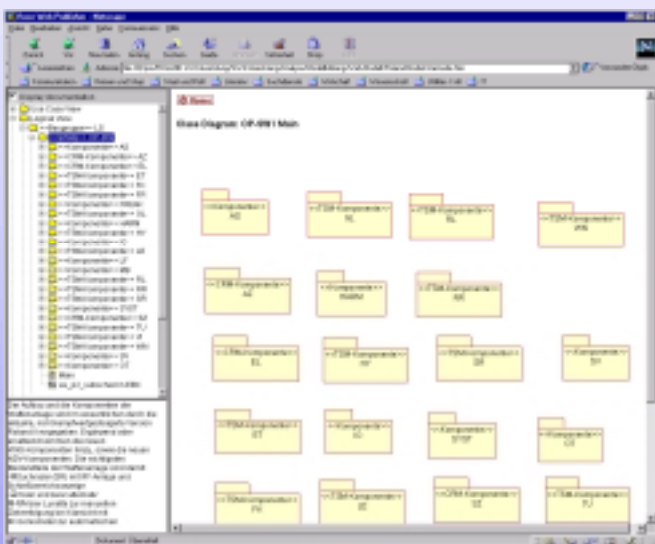
# Assessment Konsequenzen

- **Definition einer Analyse-, Umsetzungs- und Nachweisphase**
- **Einführung eines Fehlertrackings**
- **Harmonisierung der Systemdefinitionen durch Einsatz eines UML-Modells**
- **Sanfte Verbesserung der SW-Architektur**
- **Definition eines Nachweiskonzeptes auf Basis der Fehlertrackingergebnisse unter Verwendung des UML-Modells**

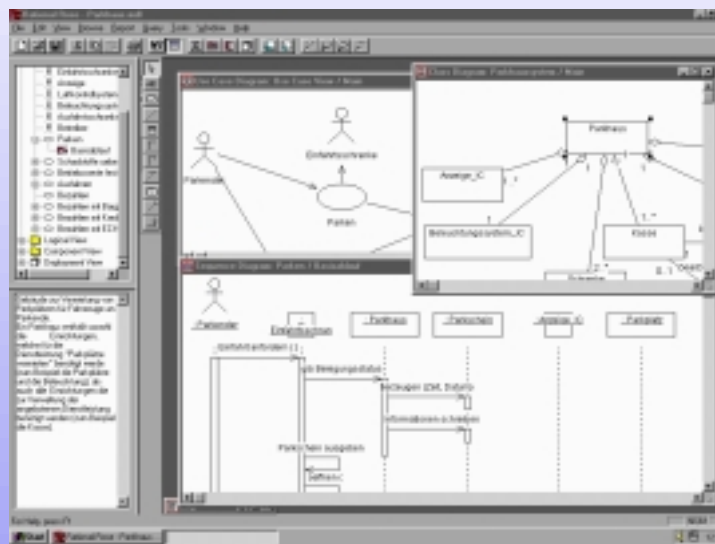
# Motivation für Modellbildung

- **Komplexität des Systems**
  - **Grosses System besteht aus n Anwendungssystemen**
  - **Entwicklung/Integration unterschiedlicher Anwendungssysteme, HW, Betriebssysteme, Netzwerktechnologien**
  - **Konsistenz zwischen Teilsystemen**
  - **Beherrschung der Abhängigkeiten**
- **Schaffung einer zentralen Wissensbasis**
- **Generierung von Anwendungsdokumenten**
- **Hilfsmittel für Nachweis/Test**

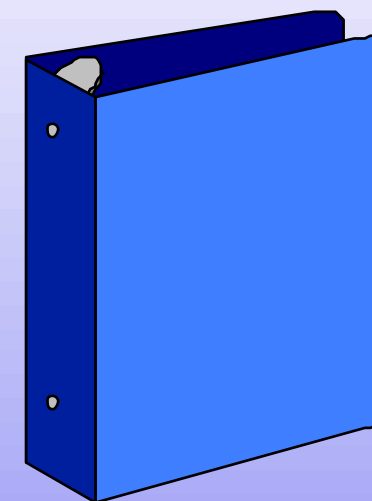
# Modell - Arbeit im Team



**Web Modell**  
Roland-Model-Startseite.htm



**Rose Modell**  
Roland-LS.mdl

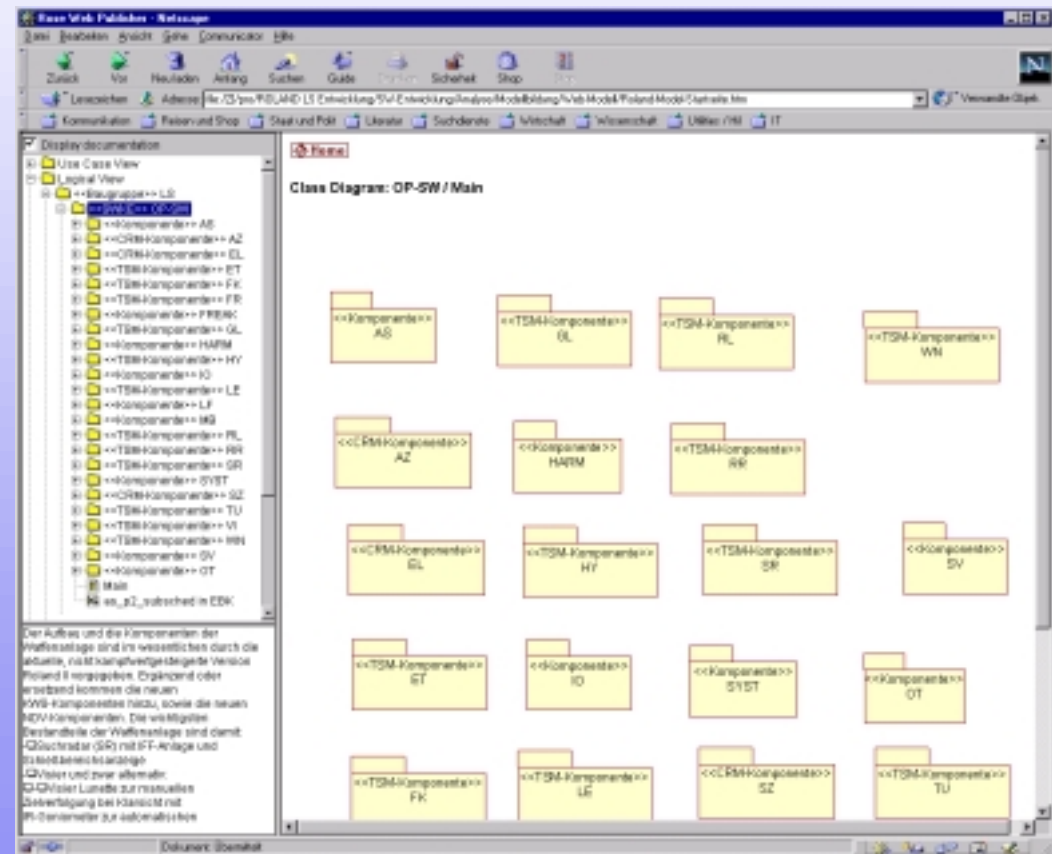


**Modell**  
Handbuch.doc



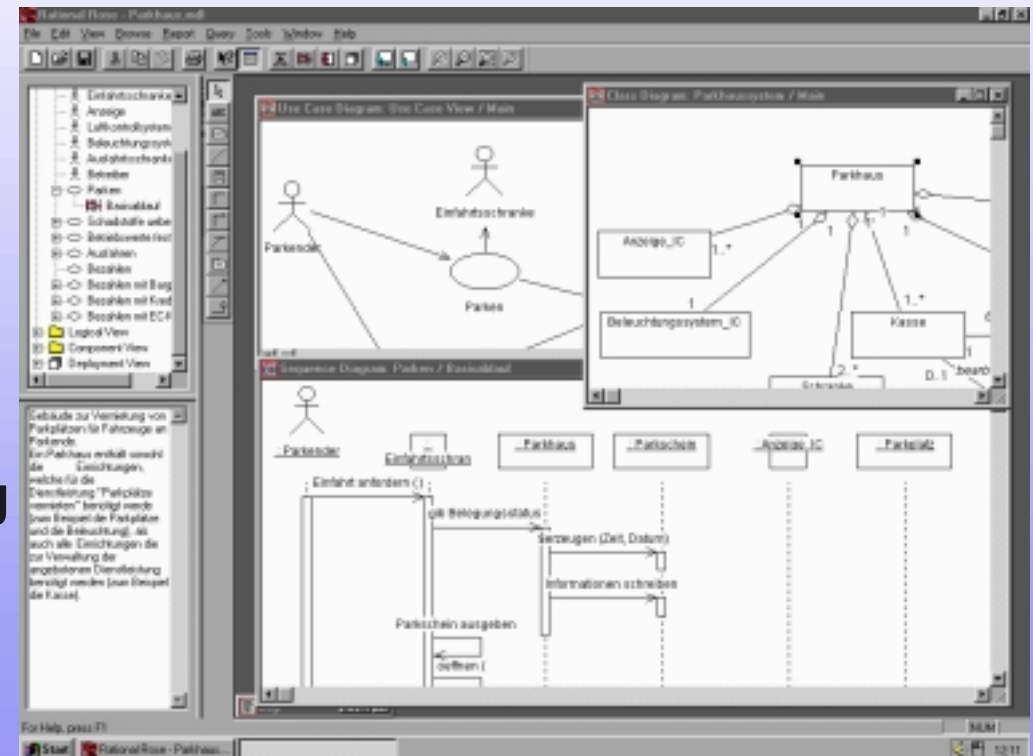
# Web Modell

- dient als Info-Quelle
- nur Browser nötig
- kein Zusatzkönnen nötig
- read only
- drei Frames
  - Modell-Browser
  - Detailinfo-Bereich
  - Dokumentationsbereich



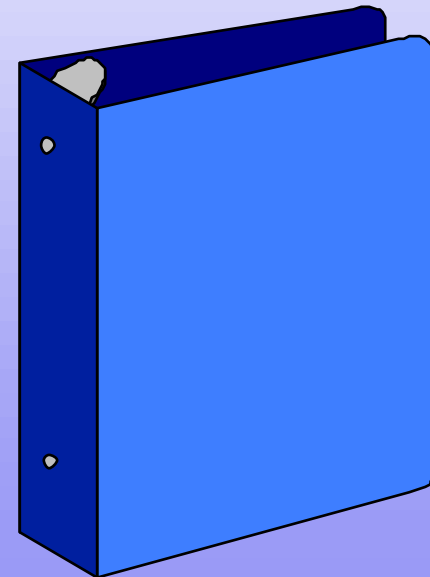
# Rose Modell

- zur Modellerstellung
- Rational Rose nötig
- Rose Kenntnisse nötig
- zentraler Master
  - Dokumentationsgenerierung
  - Testfallgenerierung
  - Konsistenzchecks
  - Web-Modell-Generierung



# Roland LS Modell Handbuch

- **Bedienungsanweisung zum Modell**
- **Word-File**
- **dreiteilige Erläuterung**
  - **Modellaufbau**
  - **Modellerstellung,  
Modellierungsrichtlinien**
  - **Dokumentationsgenerierung,  
Testfallgenerierung,  
Konsistenzchecks**

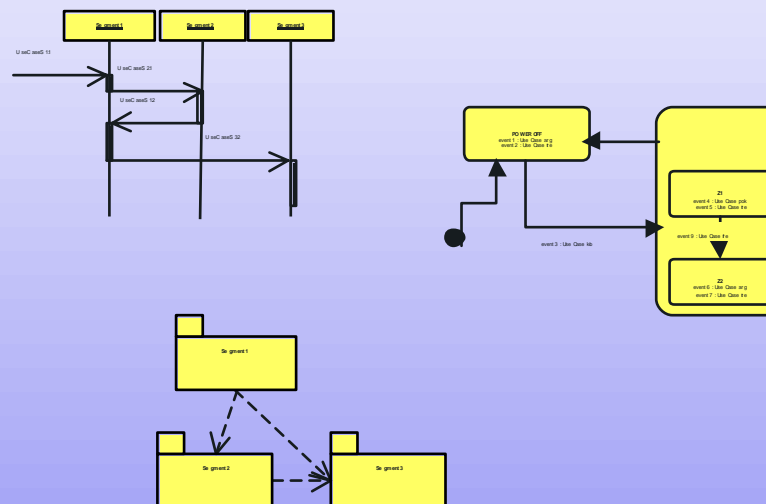


**Modellhandbuch.doc**

# Roland-LS-Modellierung

- **Verwendung von Standardmodellierungsmittel**

- Ablaufdiagramme
- Zustandsdiagramme
- Aktivitätsdiagramme
- Strukturierungsmittel



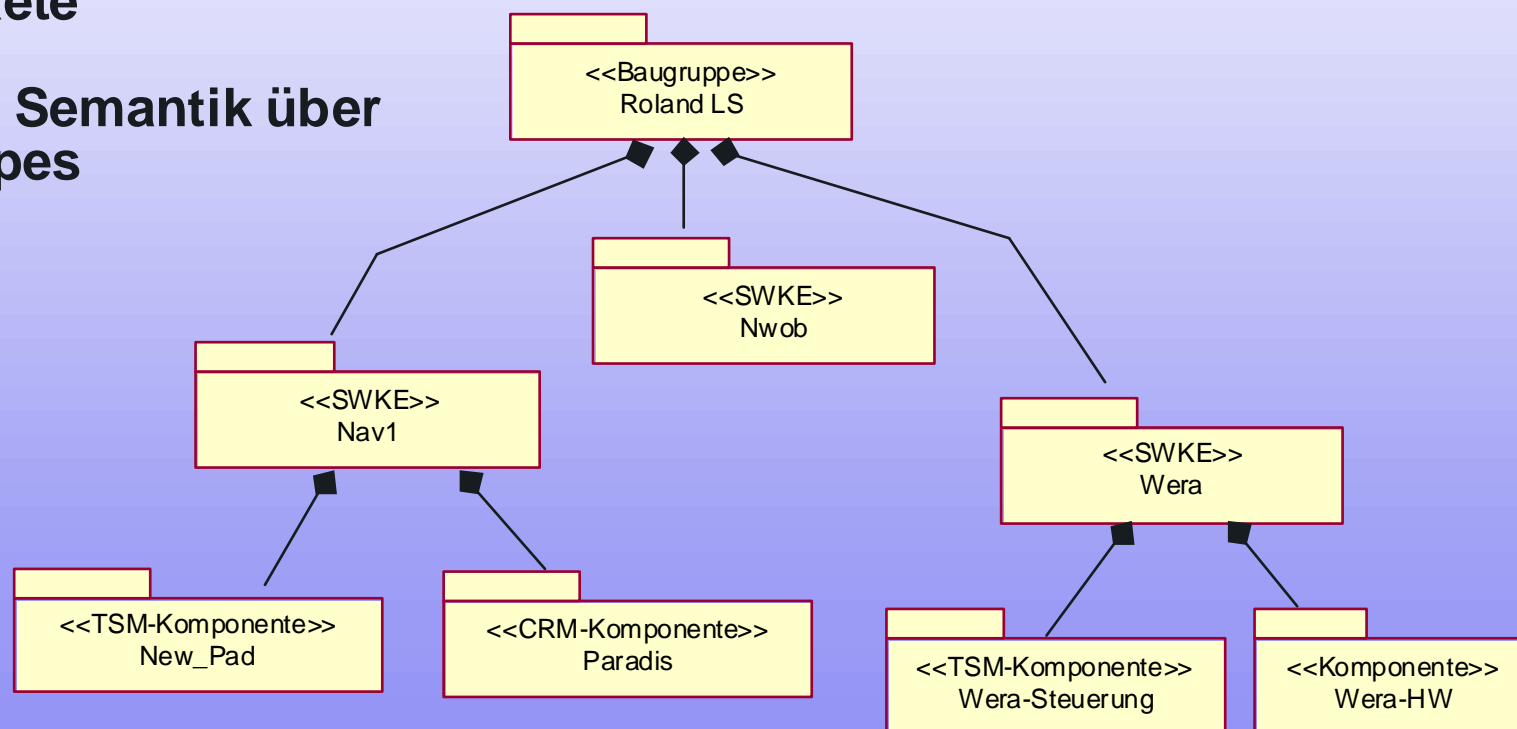
- **Ziele**

- **Zentrale Wissensbasis**
- **Beherrschung der Komplexität**
- **Erfassung der Beziehungen und Abhängigkeiten (Tracebarkeit)**

# Roland-LS-Modellierung

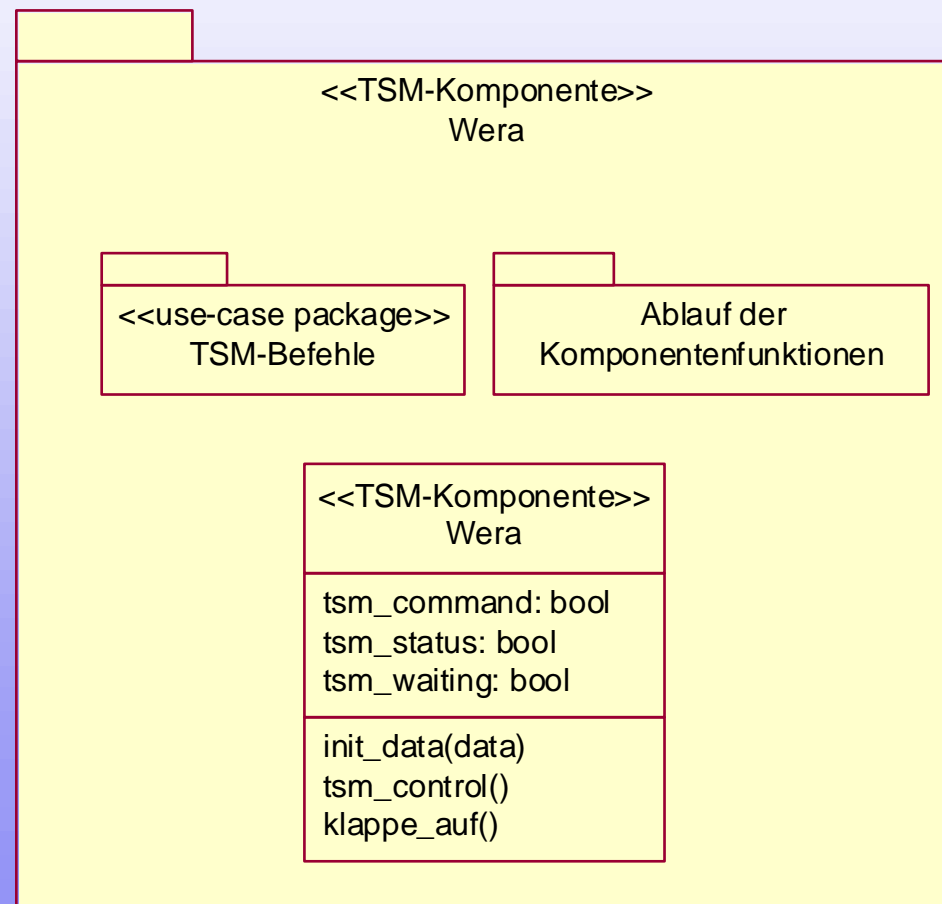
- Systemstrukturierung durch

- UML Pakete
- V-Modell Semantik über Stereotypes



# Roland-LS-Modellierung

- UML Pakete für
  - Anforderungen
  - Interne Funktionsabläufe
  
- Klasse
  - Attribute für Variable
  - Operationen für Funktionen



# Roland-LS-Modellierung

- **MilBus**

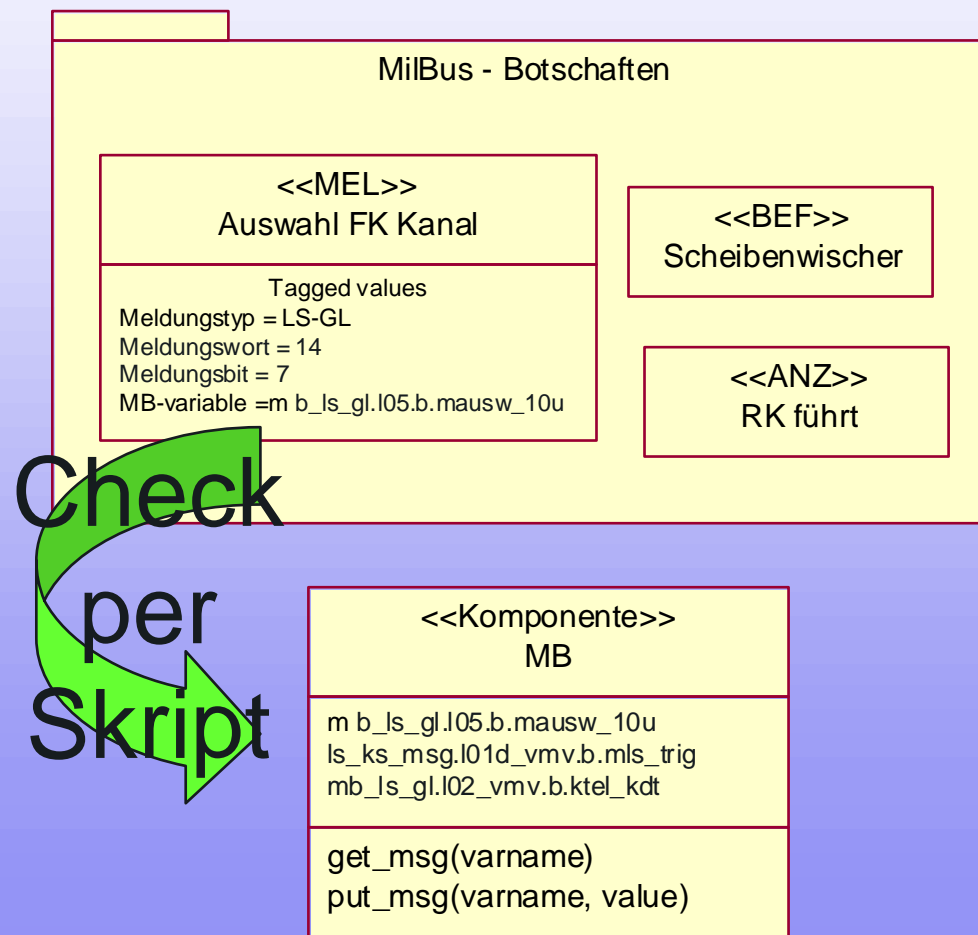
- **Botschaften als Klassen**

- **weitere Informationen über tagged values**

- zugehörige Variable
- Meldungswort
- ....

- **MB-Treiber**

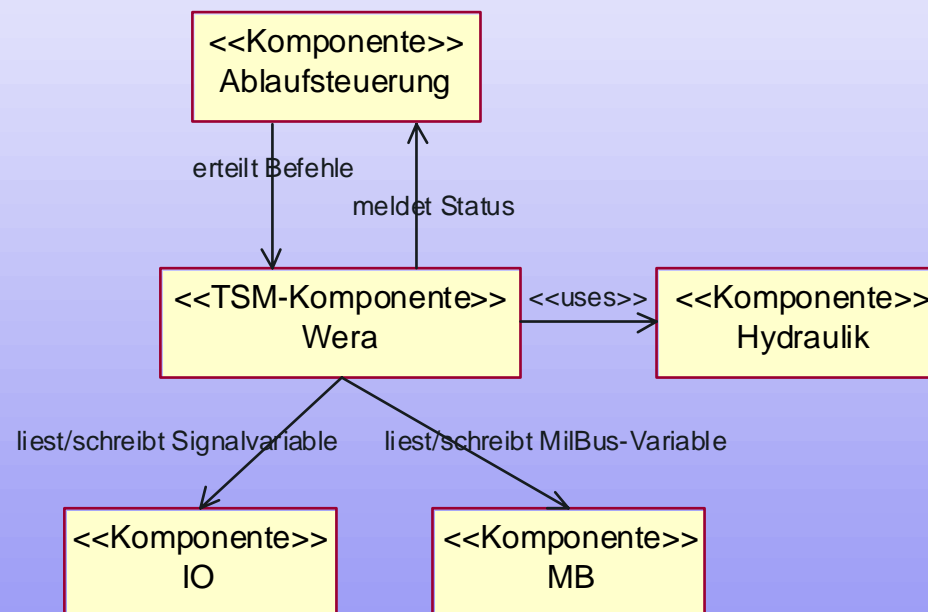
- **MB-Variable**



# Roland-LS-Modellierung

- **Schnittstellenmodellierung**

- **Klasse**
- **Assoziationen**
- **<<uses>> für Abhängigkeiten**
- **erlaubt**
  - Strukturprüfung in Interaktionen
  - Abhängigkeitsprüfung





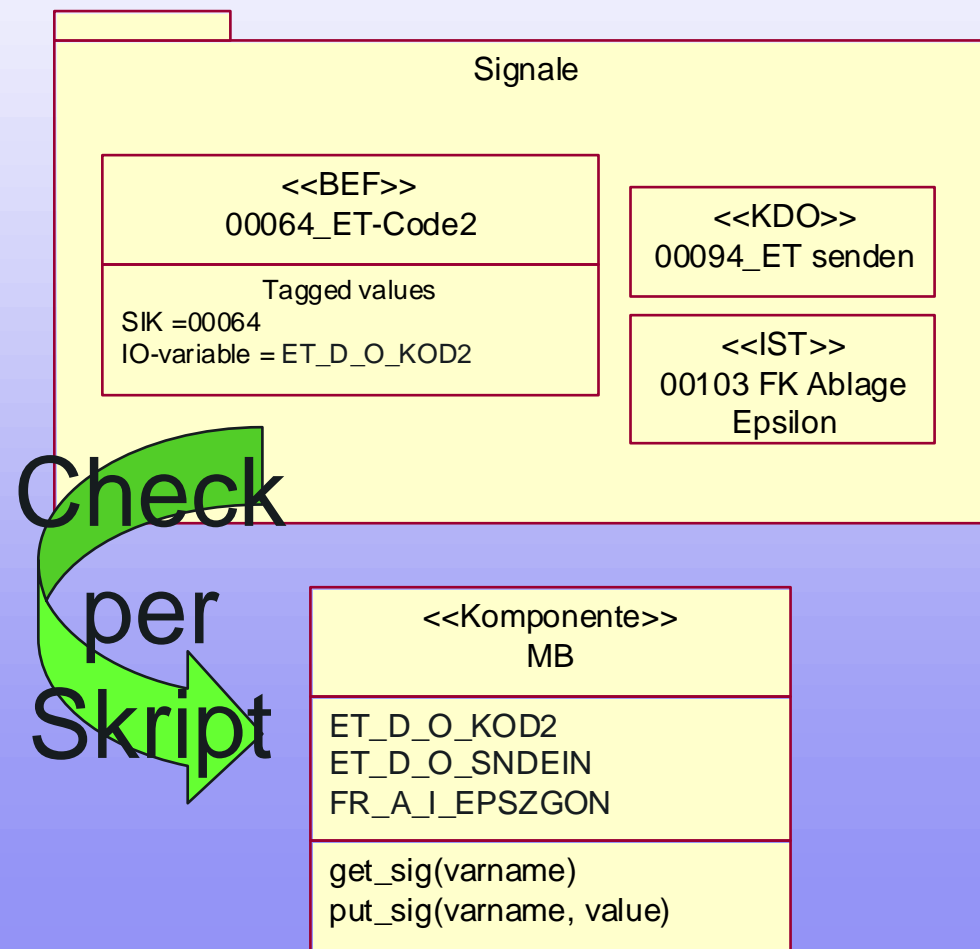
# Roland-LS-Modellierung

- **Signale**

- **Signale als Klassen**
- **weitere Informationen über tagged values**
  - SIK-Nummer
  - IO-Variable
  - ....

- **IO-Treiber**

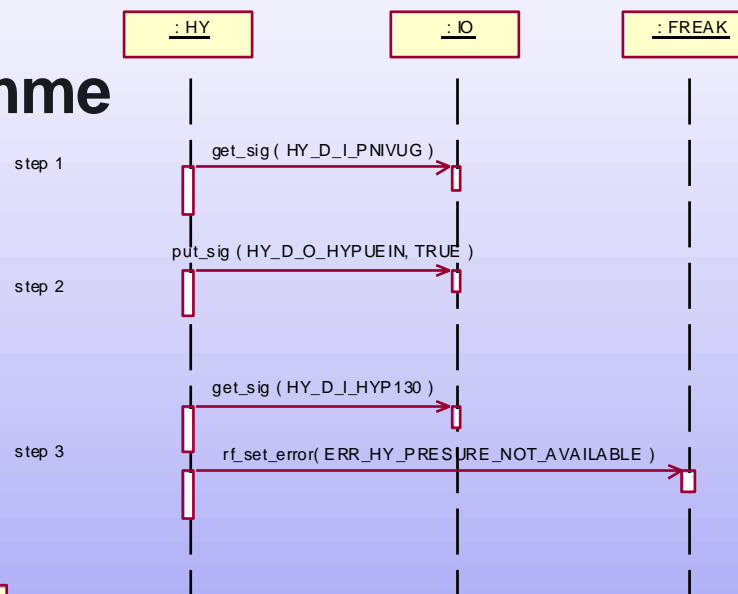
- **IO-Variable**



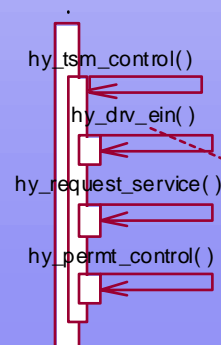
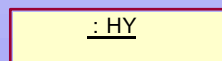
# Roland-LS-Modellierung

- Abläufe durch Sequenzdiagramme

- Funktionsaufrufe
- Signalzugriffe
- Milbus-Zugriffe



- Verschachtelte Aufrufe

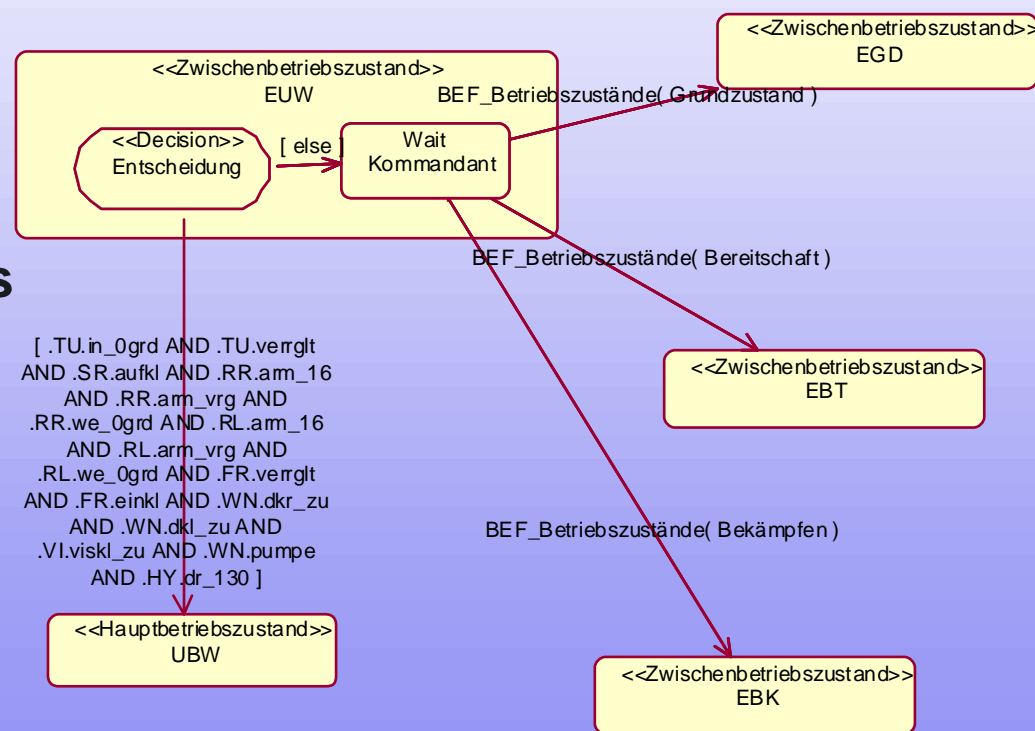
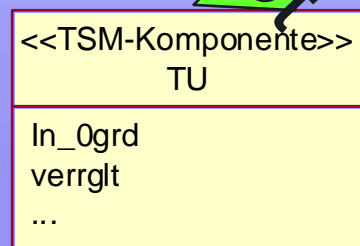


Sequence Diagram: Signalablauf der Komponenten/ hy\_drv\_ein - Basisablauf intern

# Roland-LS-Modellierung

- Systemzustände durch Statecharts

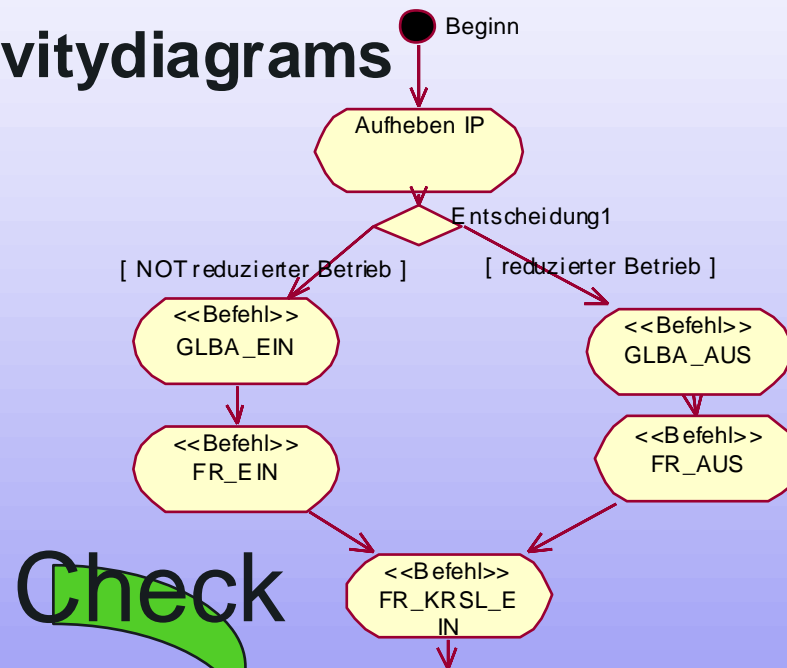
- Zustände
- Signale als Ereignisse
- Geschachtelte Statecharts
- Bedingungen mit Komponentenvariablen



# Roland-LS-Modellierung

- Systemaktivitäten durch Activitydiagrams

- Aktivitäten
- Bedingungen
- Abgleich mit TSM-Befehlen



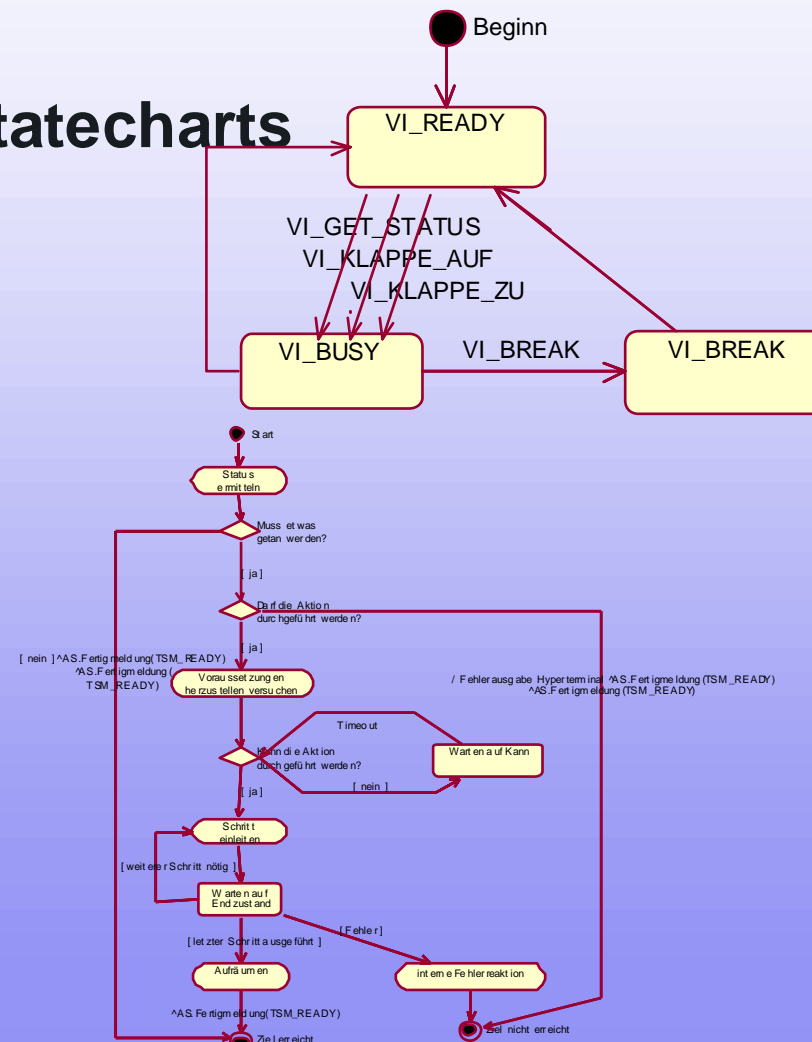
Check  
per  
Skript

<<TSM-Komponente>> FR
Befehle FR_EIN FR_AUS

# Roland-LS-Modellierung

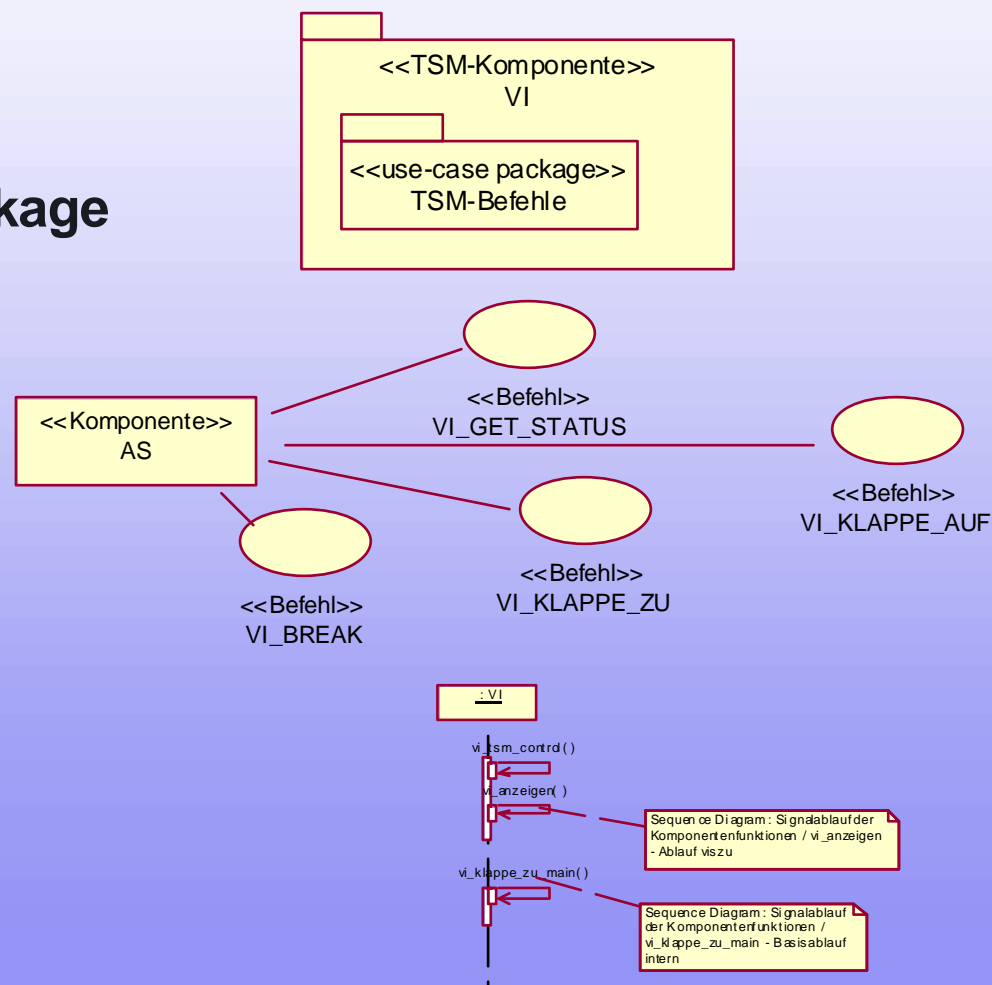
- TSM-Komponenten durch Statecharts

- Zustände
- TSM-Befehle als Ereignisse
- Bedingungen
- Interne Aktivitäten
- Geschachtelte Statecharts



# Roland-LS-Modellierung

- **TSM-Befehle**
  - gruppiert in Use Case Package
  
- Befehle als Use Cases
- Externe Komponenten als Aktoren (Schnittstellen!)
  
- Interne Abläufe über Sequenzdiagramme

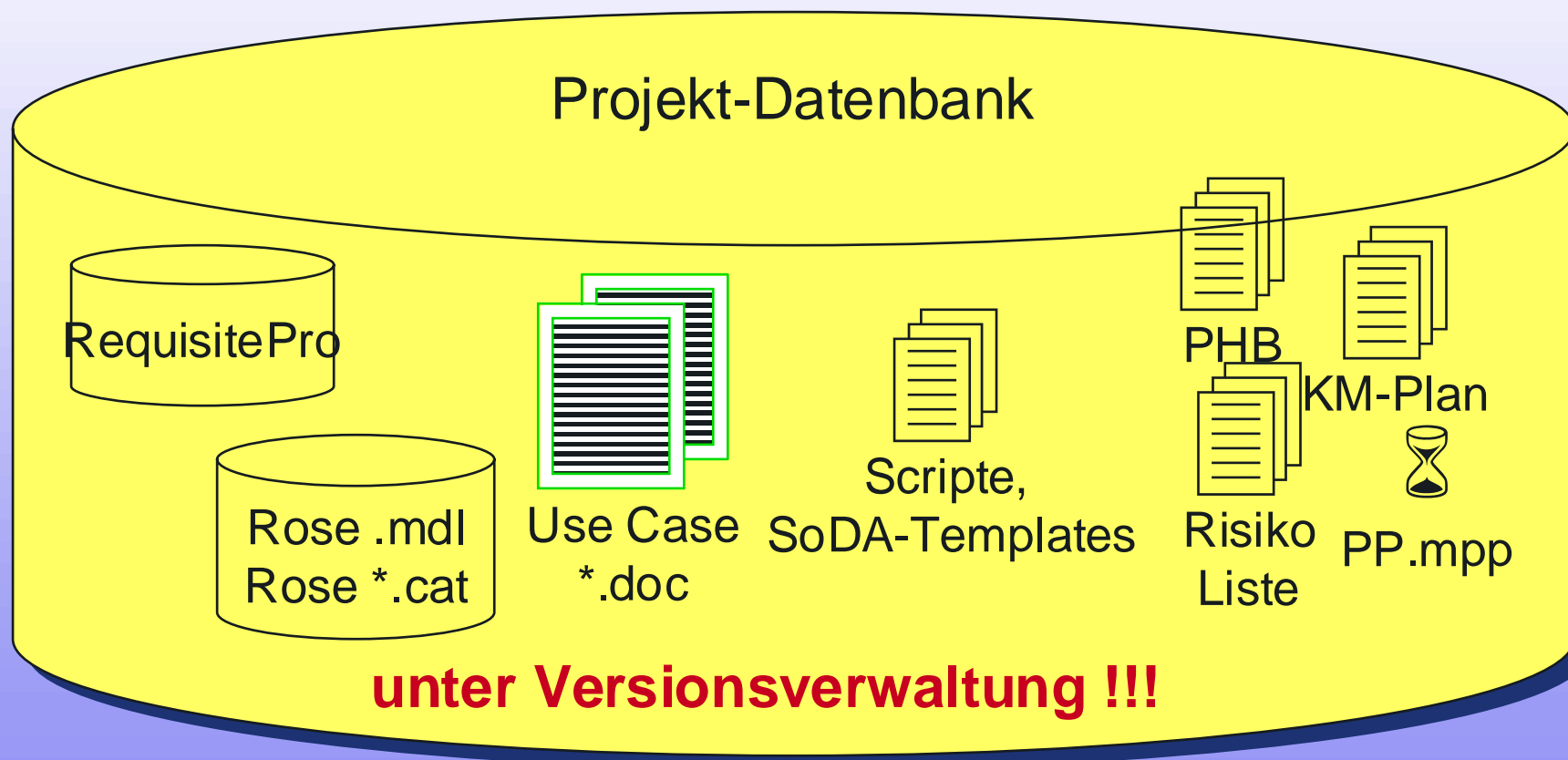


# Modellierung mit Rose

## Wieso Rational Rose?

- **Offenes, einfaches API**
  - Skriptgestützte (halbautomatische) Modellerstellung aus Excelfiles
  - beliebige Konsistenzprüfungen
- **Flexible Dokumentationsgenerierung**
  - automatische Generierung V-Modell konformer Dokumente
- **Einfache Handhabbarkeit mit WebPublishing**
  - Kurze Einarbeitung, geringe Lizenzkosten
- **Erweiterbarkeit durch UML-Standard**
  - Simulation

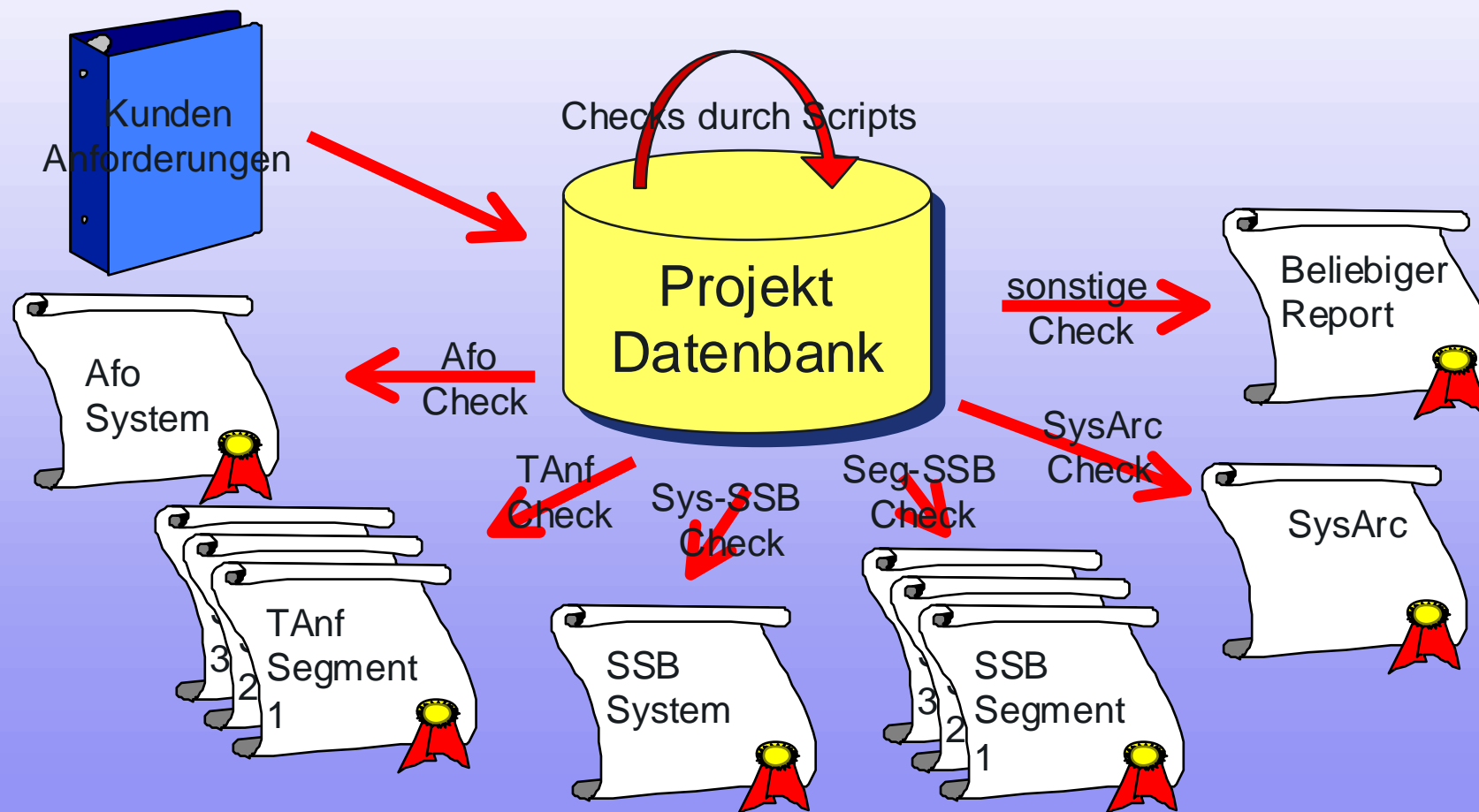
# Lösungsansatz für Projektdatenbank





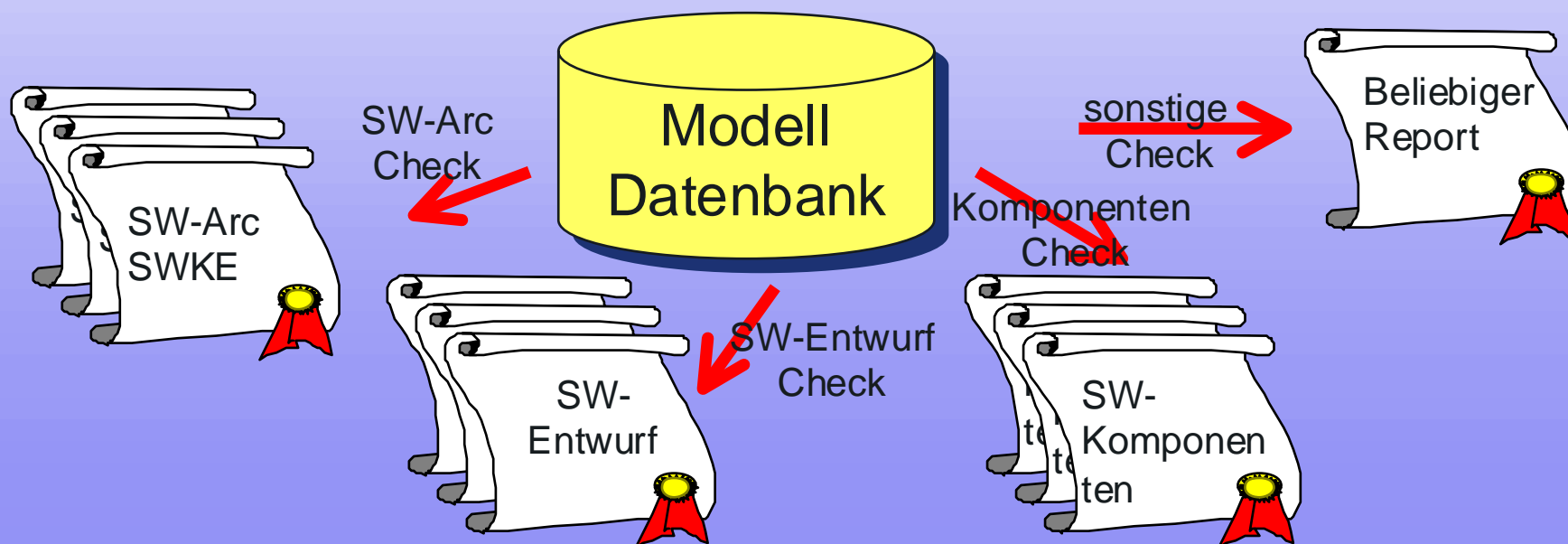
# Automatische Dokumenten Generierung

Konsistenzchecks und Dokumentengenerierung



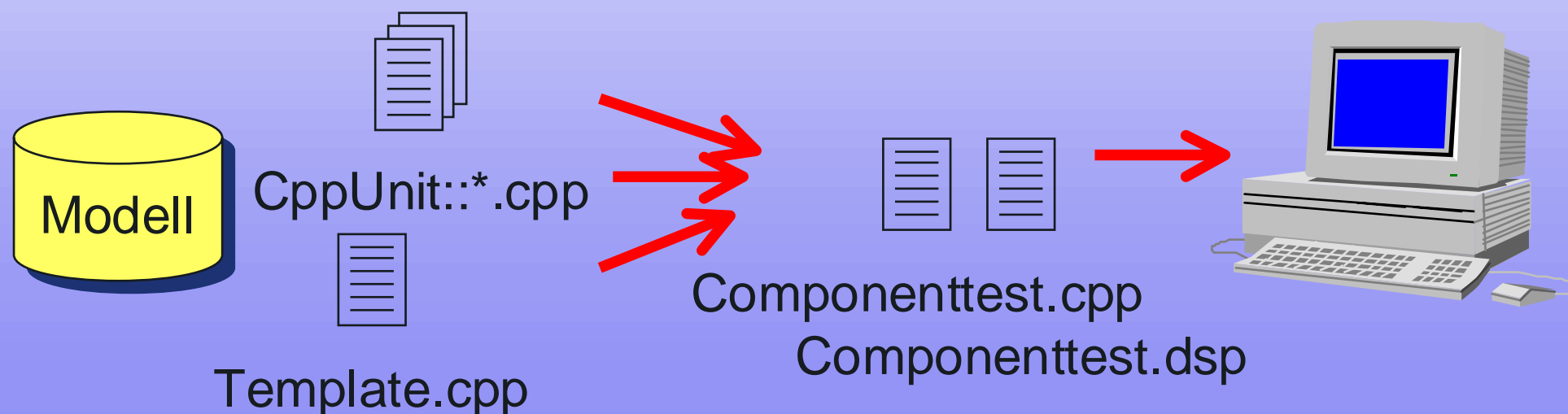
# Automatische Dokumenten-Generierung

- **Script-basierte Konsistenzchecks**
  - Aufdeckung von Entwurfsfehlern
- **Templatebasierte Dokumentengenerierung**
  - Drastische Reduzierung des Dokumentationsaufwandes



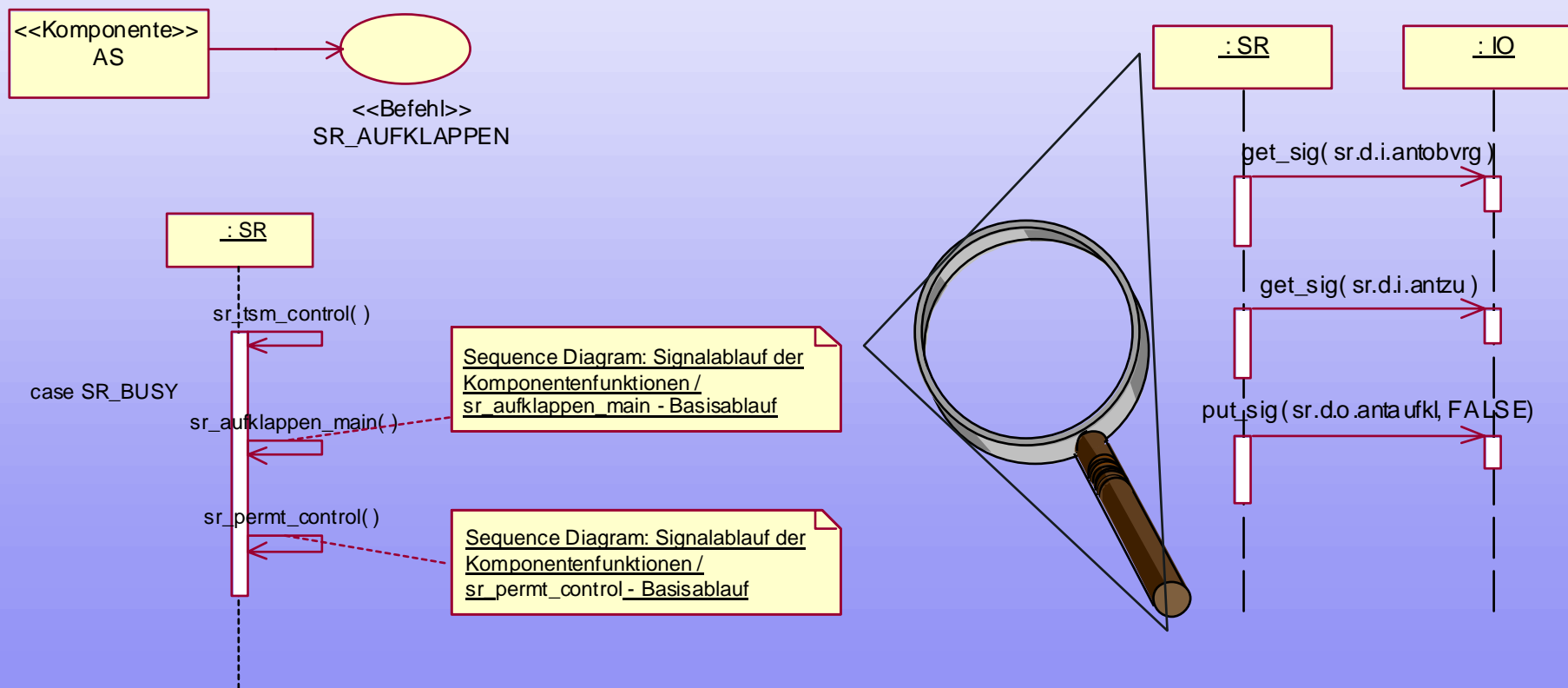
# Testfallgenerierung, Simulation

- **Testfallgenerierung für Komponenten-Tests**
  - scriptbasiert Generierung der Testframes
  - Template basiert
  - basierend auf CppUnit
  - Regressionstest fähig



# ROSE Modellierung

## Komponentenbefehle

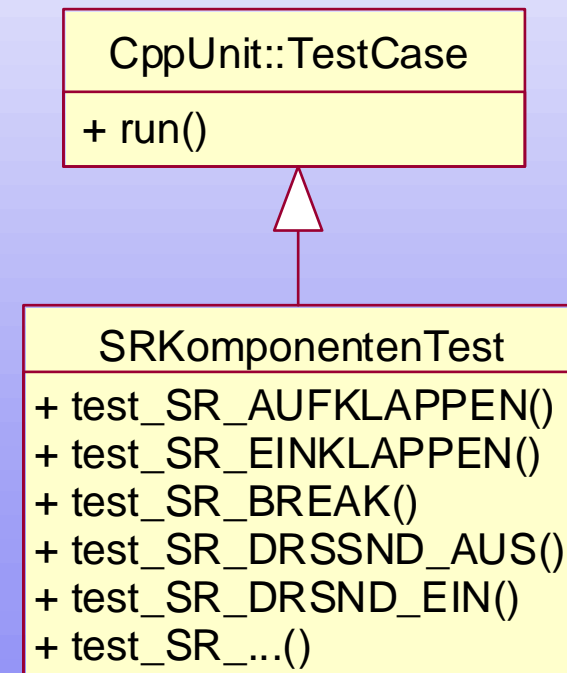


# Testkonzept

Testfälle an Komponenteninterface orientiert

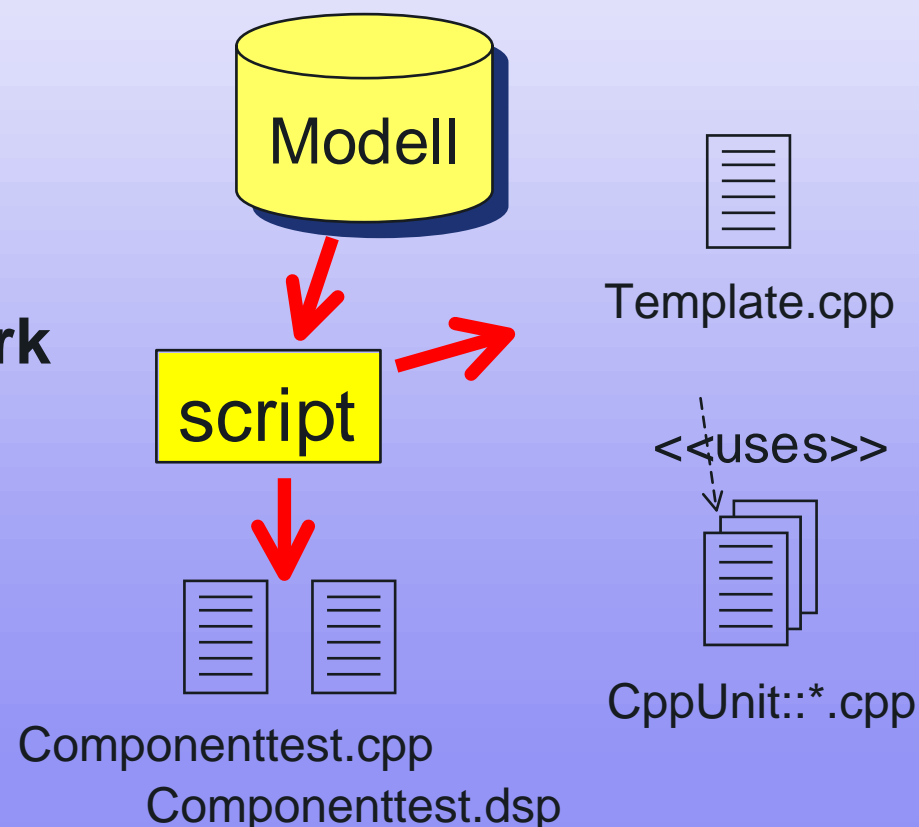
- Komponentenbefehlen sind Interface
- Tests basierend auf Basisklasse
- Ausblendung von Testfällen möglich
- Vor-/Nachbedingungen durch Bus-/Signalin-/outputs

Generierbar aus Modell



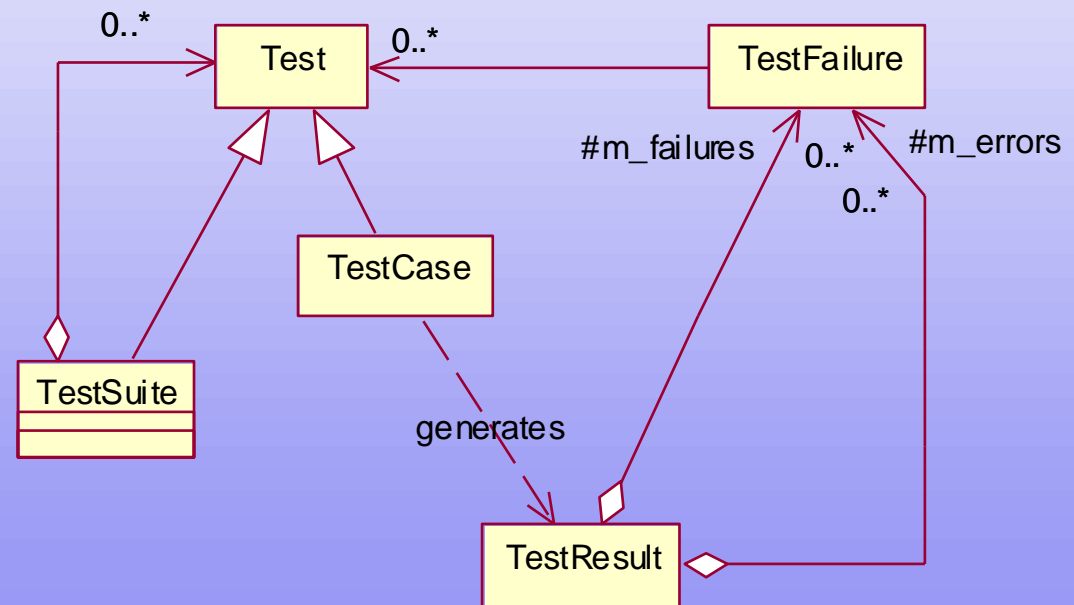
# Testfallgenerierung

- **Script extrahiert Modellinformationen**
  - Generierung unabhängig von Modelländerungen
- **Script füllt Template**
  - Leichte Erweiterbarkeit
- **Template nutzt Testframework**
  - Ausgetestetes CppUnit
- **Regressionsfähigkeit**
  - Integration in TestManager



# Testfallgenerierung

- Testframework CppUnit
  - analog zu Junit für C++
  - open source
  - stellt Basis-funktionalität zu Verfügung



# Typischer Testcode

```
// test code for command SR::SR_EINKL
public:
void test_SR_EINKL( ) {
    int i;
    // include here test code for command SR::SR_EINKL

    // set preconditions and pre-requisties for the test

    /* must condition: sr_tsm_sw.b.einkl == 0 */
    sr_tsm_sw.b.eingeklappt = 0;

    /* need-condition: HY-Druck vorhanden AND sr_tsm_sw.b.ip_klapp == 0 */
    sr_tsm_sw.b.ip_klapp = 0;

    sr_tsm_ctrl = SR_EINKL;

    sr.d.i.grd0 = TRUE;

    // run SR_EINKL 10 cycles and waiting
    for ( i=1; i<10; i++ ) ::sr_tsm_control();

    // wait 3 sec. And receive answer
    sleep(3000);
    sr.d.i.antobvrg = FALSE;

    for ( i=1; i<10; i++ ) ::sr_tsm_control();

    // check expected results against real results
    CPPUNIT_ASSERT(sr_tsm_ctrl == SR_READY);
    // CPPUNIT_ASSERT(sr_tsm_sw.b.eingeklappt == 1);
    CPPUNIT_ASSERT(sr_tsm_sw.b.aufgeklappt == 0);
    // CPPUNIT_ASSERT(mb_ls_ks.l01d_vmv.b.ksr_senderl == 0);
}
}
```

*generierter Code  
händische Auswahl der  
Testinput/-output*



# Testtemplate

```
//+-----
//|                               File Name: <COMPONENT_NAME>KomponentenTest
//+-----

// includes for CppUnit
#include <cppunit/TestSuite.h>

// include for component to test
extern "C" {
#include "op/<component_name>x.h"
#include "op/io.h"
#include "op/mb.h"
#include "../simul.h"
}

class <test_name> : public CppUnit::TestCase {
public:
void setUp () {
// initialize here objects for the test fixture.
}
// <command_tests> code for each command is included here using the template TSMBefehlTestTemplate.cpp

// define a suite of test cases

<test_name>() : TestCase("This is the component test for component <COMPONENT_NAME>") {
// add here any additional constructor code
}

public:
static CppUnit::TestSuite *suite () {
CppUnit::TestSuite *suiteOfTests = new CppUnit::TestSuite("Test suite for component <COMPONENT_NAME>");
// each command test is added here to the <test_suite>. See the script generateKomponentenTestCode.ebs
return suiteOfTests;
}
};
```

# Fragen / Kontakte

**Thomas Rittel, EADS/LFK**  
**Thomas.Rittel@lfk.eads.net**

**Dr. Rudolf Hauber, Kölsch & Altmann GmbH**  
**rhauber@ka-muc.de**